Catalyst 3850 Series 스위치의 높은 CPU 사용량 문제 해결

목차

소개

이 문서에서는 새 Cisco IOS® XE 플랫폼에서 주로 인터럽트로 인한 CPU 사용량 문제를 해결하는 방법에 대해 설명합니다.

배경 정보

Cisco IOS® XE의 구축 방식을 이해하는 것이 중요합니다. Cisco IOS® XE를 통해 Cisco는 Linux 커널로 전환했으며, 모든 하위 시스템은 프로세스로 나누어졌습니다. 모듈 드라이버, 고가용성(HA) 등 이전에 Cisco IOS 내부에 있던 모든 하위 시스템이 이제 Linux 운영 체제(OS) 내에서 소프트웨어 프로세스로 실행됩니다. Cisco IOS 자체는 Linux OS(IOSd) 내에서 데몬으로 실행됩니다.Cisco IOS® XE는 기존 Cisco IOS®와 동일한 모양과 느낌뿐만 아니라 운영, 지원 및 관리도 유지합니다.

또한 이 문서에서는 CPU 사용량 문제를 해결하기 위해 필수적인 이 플랫폼의 여러 새 명령을 소개합니다.

몇 가지 유익한 정의를 소개합니다.

- FED(Forwarding Engine Driver): Cisco Catalyst 3850 Series 스위치의 핵심으로서 모든 하드웨어 프로그래밍/포워딩을 담당합니다.
- Cisco IOSd: Linux 커널에서 실행되는 Cisco IOS® 데몬입니다. 커널 내에서 소프트웨어 프로세스의 형태로 실행됩니다.
- PDS(Packet Delivery System): 다양한 하위 시스템에 패킷이 전달되고 그 하위 시스템에서 패킷이 전달되는 방식을 보여 주는 아키텍처 및 프로세스입니다. 예를 들면, FED에서 IOSd로 또는 그 반대로 패킷이 전달되는 방법을 제어합니다.

- 핸들: 핸들은 포인터로 간주될 수 있습니다. 이 핸들을 통해 시스템에서 생성하는 출력에 쓰이는 특정 변수에 관한 세부 정보를 찾을 수 있습니다. Cisco Catalyst 6500 시리즈 스위치에 있는 LTL(Local Target Logic) 색인의 개념과 비슷합니다.

사례 연구: ARP(Address Resolution Protocol) 인터럽트

이 섹션의 문제 해결 및 확인 프로세스는 인터럽트로 인한 높은 CPU 사용량에 대해 광범위하게 적용할 수 있습니다.

1단계: CPU 사이클을 소비하는 프로세스 파악

물론 show process cpu 명령은 CPU의 현재 상황을 보여줍니다. Cisco Catalyst 3850 Series Switch는 4개의 코어를 사용하며, 4개의 코어 모두에 대해 나열된 CPU 사용량을 확인할 수 있습니다.

<#root>

3850-2#

show processes cpu sorted | exclude 0.0

```
Core 0: CPU utilization for five seconds: 53%; one minute: 39%;  five minutes: 41%
Core 1: CPU utilization for five seconds: 43%; one minute: 57%;  five minutes: 54%
Core 2: CPU utilization for five seconds: 95%; one minute: 60%;  five minutes: 58%
Core 3: CPU utilization for five seconds: 32%; one minute: 31%;  five minutes: 29%
PID    Runtime(ms) Invoked  uSecs  5Sec     1Min     5Min     TTY   Process
8525   472560      2345554  7525   31.37    30.84    30.83    0     iosd
5661   2157452     9234031  698    13.17    12.56    12.54    1088  fed
6206   19630       74895    262    1.83     0.43     0.10     0     eicored
6197   725760      11967089 60     1.41     1.38     1.47     0     pdsd
```

출력에서 Cisco IOS® 데몬은 이 상자의 핵심인 FED와 함께 CPU의 주요 부분을 소비합니다. 인터럽트로 인해 CPU 사용량이 많은 경우 Cisco IOSd 및 FED가 CPU의 주요 부분을 사용하고 이러한 하위 프로세스(또는 하위 프로세스)가 CPU를 사용하는 것을 확인할 수 있습니다.

- FED Punject TX
- FED Punject RX
- FED Punject replenish
- FED Punject TX complete

show process cpu detailed <process> 명령을 사용하여 이 프로세스를 자세히 들여다볼 수 있습니다. Cisco IOSd가 CPU 사용량의 대부분을 담당하므로 이에 대해 자세히 알아보겠습니다.

<#root>

3850-2#

show processes cpu detailed process iosd sorted | ex 0.0

```
Core 0: CPU utilization for five seconds: 36%; one minute: 39%; five minutes: 40%
Core 1: CPU utilization for five seconds: 73%; one minute: 52%; five minutes: 53%
Core 2: CPU utilization for five seconds: 22%; one minute: 56%; five minutes: 58%
Core 3: CPU utilization for five seconds: 46%; one minute: 40%; five minutes: 31%
PID    T C  TID   Runtime(ms)Invoked uSecs 5Sec   1Min   5Min  TTY  Process
                                          (%)    (%)    (%)
8525   L          556160     2356540 7526  30.42   30.77  30.83  0   iosd
8525   L 1  8525  712558     284117  0     23.14   23.33  23.38  0   iosd
59     I          1115452    4168181 0     42.22   39.55  39.33  0   ARP Snoop
198    I          3442960    4168186 0     25.33   24.22  24.77  0   IP Host Track Proce
30     I          3802130    4168183 0     24.66   27.88  27.66  0   ARP Input
283    I          574800     3225649 0     4.33    4.00   4.11   0   DAI Packet Process

3850-2#
```

show processes cpu detailed process fed sorted | ex 0.0

```
Core 0: CPU utilization for five seconds: 45%; one minute: 44%; five minutes: 44%
Core 1: CPU utilization for five seconds: 38%; one minute: 44%; five minutes: 45%
Core 2: CPU utilization for five seconds: 42%; one minute: 41%; five minutes: 40%
Core 3: CPU utilization for five seconds: 32%; one minute: 30%; five minutes: 31%
PID    T C  TID   Runtime(ms)Invoked uSecs 5Sec   1Min   5Min   TTY  Process
                                          (%)    (%)    (%)
5638   L          612840     1143306 536   13.22  12.90   12.93  1088  fed
5638   L 3  8998  396500     602433  0     9.87   9.63    9.61   0     PunjectTx
5638   L 3  8997  159890     66051   0     2.70   2.70    2.74   0     PunjectRx
```

출력(Cisco IOSd CPU 출력)은 ARP Snoop, IP Host Track Process 및 ARP Input이 높음을 보여줍니다. CPU가 ARP 패킷 때문에 인터럽트될 때 흔히 볼 수 있는 현상입니다.

2단계: CPU 사용량이 많은 상태를 유발하는 CPU 큐 확인

Cisco Catalyst 3850 시리즈 스위치에는 수많은 대기열이 있어 각기 다른 패킷 유형을 다룹니다. FED에서 32개의 RX CPU 대기열을 유지 관리하는데, 이는 CPU로 직접 연결되는 대기열입니다. 어떤 패킷이 CPU로 전송되며 Cisco IOSd에서 처리되는지 알아보려면 이러한 큐를 모니터링하는 것이 중요합니다. 이 대기열은 ASIC을 기준으로 합니다.

✎ 참고: ASIC는 2개, 즉 0과 1입니다. 포트 1에서 24는 ASIC 0에 속합니다.

대기열을 보려면 show platform punt statistics port-asic <port-asic>cpuq <queue> direction 명령을 실행합니다.

show platform punt statistics port-asic 0 cpuq -1 direction rx 명령에서 -1 인수는 모든 대기열을 나열합니다. 따라서 이 명령은 Port-ASIC 0에 대한 모든 수신 대기열을 나열합니다.

이제 어떤 대기열에서 많은 수의 패킷을 빠른 속도로 푸시하는지를 식별해야 합니다. 이 예에서는 대기열을 조사하여 다음과 같이 원인을 밝혀냈습니다.

```
<snip>
RX (ASIC2CPU) Stats (asic 0 qn 16 lqn 16):
```

```
RXQ 16: CPU_Q_PROTO_SNOOPING
---------------------------------------
Packets received from ASIC     : 79099152
Send to IOSd total attempts    : 79099152
Send to IOSd failed count      : 1240331
RX suspend count               : 1240331
RX unsuspend count             : 1240330
RX unsuspend send count        : 1240330
RX unsuspend send failed count : 0
RX dropped count               : 0
RX conversion failure dropped  : 0
RX pkt_hdr allocation failure  : 0
RX INTACK count                : 0
RX packets dq'd after intack   : 0
Active RxQ event               : 9906280
RX spurious interrupt          : 0
<snip>
```

대기열 번호는 16이고, 대기열 이름은 CPU_Q_PROTO_SNOOPING입니다.

문제의 원인인 대기열을 찾아내는 또 다른 방법으로, show platform punt client 명령을 입력합니다.

<#root>

```
3850-2#

show platform punt client
```

```
  tag      buffer        jumbo    fallback     packets   received   failures
                                              alloc   free  bytes    conv  buf
 27       0/1024/2048     0/5       0/5         0     0          0     0     0
 65536    0/1024/1600     0/0       0/512       0     0          0     0     0
 65537    0/ 512/1600     0/0       0/512    1530  1530     244061     0     0
 65538    0/   5/5        0/0       0/5         0     0          0     0     0
 65539    0/2048/1600     0/16      0/512       0     0          0     0     0
 65540    0/ 128/1600     0/8       0/0         0     0          0     0     0
 65541    0/ 128/1600     0/16      0/32        0     0          0     0     0
 65542    0/ 768/1600     0/4       0/0         0     0          0     0     0
 65544    0/  96/1600     0/4       0/0         0     0          0     0     0
 65545    0/  96/1600     0/8       0/32        0     0          0     0     0
 65546    0/ 512/1600     0/32      0/512       0     0          0     0     0
 65547    0/  96/1600     0/8       0/32        0     0          0     0     0
 65548    0/ 512/1600     0/32      0/256       0     0          0     0     0
 65551    0/ 512/1600     0/0       0/256       0     0          0     0     0
 65556    0/  16/1600     0/4       0/0         0     0          0     0     0
 65557    0/  16/1600     0/4       0/0         0     0          0     0     0
 65558    0/  16/1600     0/4       0/0         0     0          0     0     0
 65559    0/  16/1600     0/4       0/0         0     0          0     0     0
 65560    0/  16/1600     0/4       0/0         0     0          0     0     0
s65561  421/ 512/1600     0/0       0/128   79565859 131644697  478984244     0 37467
 65563    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65564    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65565    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65566    0/ 512/1600     0/16      0/256       0     0          0     0     0
 65581    0/   1/1        0/0       0/0         0     0          0     0     0
 131071   0/  96/1600     0/4       0/0         0     0          0     0     0
fallback pool: 98/1500/1600
jumbo pool:    0/128/9300
```

패킷 대부분이 할당된 태그를 확인합니다. 이 예에서는 65561입니다.

그리고 다음 명령을 입력합니다.

```
<#root>

3850-2#

show pds tag all | in Active|Tags|65561


   Active   Client Client
     Tags   Handle Name                 TDA       SDA         FDA   TBufD       TBytD
    65561  7296672 Punt Rx Proto Snoop  79821397  79821397    0  79821397   494316524
```

이 출력은 큐가 Rx Proto Snoop임을 보여줍니다.

show platform punt client 명령의 출력에서 65561 앞의 s는 FED 핸들이 일시 중단되었고 수신 패킷의 수가 과도함을 의미합니다. s가 사라지지 않는다면 해당 대기열이 계속 멈춰 있는 것입니다.

3단계: CPU로 전송된 패킷 덤프

show pds tag all 명령의 결과를 보면, 핸들 7296672가 Punt Rx Proto Snoop 옆에 있습니다.

show pds client <handle> packet last sink 명령에 이 핸들을 사용합니다. 이 명령을 사용하기 전에 debug pds pktbuf-last를 활성화해야 합니다. 그렇지 않으면 다음 오류가 발생합니다.

```
<#root>

3850-2#

show pds client 7296672 packet last sink


% switch-2:pdsd:This command works in debug mode only. Enable debug using
"debug pds pktbuf-last" command
```

이 디버그가 활성화되어 있으면 다음 출력이 나타납니다.

```
<#root>

3850-2#

show pds client 7296672 packet last sink


Dumping Packet(54528) # 0 of Length 60
-----------------------------------------
Meta-data
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0010 00 00 16 1d 00 00 00 00 00 00 00 00 55 5a 57 f0  ............UZW.
0020 00 00 00 00 fd 01 10 df 00 5b 70 00 00 10 43 00  .........[p...C.
0030 00 10 43 00 00 41 fd 00 00 41 fd 00 00 00 00 00  ..C..A...A......
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050 00 00 00 3c 00 00 00 00 00 01 00 19 00 00 00 00  ...<............
0060 01 01 b6 80 00 00 00 4f 00 00 00 00 00 00 00 00  .......O........
0070 01 04 d8 80 00 00 00 33 00 00 00 00 00 00 00 00  .......3........
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0090 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00a0 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 00  ................
Data
0000 ff ff ff ff ff ff aa bb cc dd 00 00 08 06 00 01  ................
0010 08 00 06 04 00 01 aa bb cc dd 00 00 c0 a8 01 0a  ................
0020 ff ff ff ff ff ff c0 a8 01 14 00 01 02 03 04 05  ................
0030 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11              ............
```

이 명령은 이 예에서 Cisco IOSd인 싱크에서 마지막으로 수신한 패킷을 덤프합니다. 여기를 보면 헤더를 덤프하며, TShark(Terminal-based Wireshark)로 디코딩 가능합니다. Meta-data는 시스템 내부에서 사용하지만, Data 출력은 실제 패킷 정보를 제공합니다. 그러나 Meta-data는 여전히 매우 유용합니다.

0070으로 시작하는 라인을 확인합니다. 아래에 표시된 것처럼, 그 다음의 첫 16비트를 사용합니다.

<#root>

3850-2#

show platform port-asic ifm iif-id 0x0104d88000000033

```
Interface Table
Interface IIF-ID        : 0x0104d88000000033
Interface Name          : Gi2/0/20
Interface Block Pointer : 0x514d2f70
Interface State         : READY
Interface Stauts        : IFM-ADD-RCVD, FFM-ADD-RCVD
Interface Ref-Cnt       : 6
Interface Epoch         : 0
Interface Type          : ETHER
        Port Type         : SWITCH PORT
        Port Location     : LOCAL
Slot              : 2
        Unit              : 20
        Slot Unit         : 20
        Acitve            : Y
        SNMP IF Index     : 22
        GPN               : 84
        EC Channel        : 0
        EC Index          : 0
        ASIC

              :

0
        ASIC Port         : 14
        Port LE Handle    : 0x514cd990
Non Zero Feature Ref Counts
        FID : 48(AL_FID_L2_PM), Ref Count : 1
```

```
        FID : 77(AL_FID_STATS), Ref Count : 1
        FID : 51(AL_FID_L2_MATM), Ref Count : 1
        FID : 13(AL_FID_SC), Ref Count : 1
        FID : 26(AL_FID_QOS), Ref Count : 1
Sub block information
        FID : 48(AL_FID_L2_PM), Private Data &colon; 0x54072618
        FID : 26(AL_FID_QOS), Private Data &colon; 0x514d31b8
```

문제의 원인이 된 인터페이스가 여기서 식별됩니다. Gig2/0/20에는 ARP 트래픽을 펌프하는 트래픽 생성기가 있습니다. 이를 종료하면 문제가 해결되고 CPU 사용량이 최소화됩니다.

4단계: FED 추적 사용

마지막 절에서 논의한 방법의 유일한 단점은 싱크대에 들어가는 마지막 패킷을 버림으로써 그것이 범인이 될 수 없다는 것이다.

이 문제를 해결할 더 나은 방법은 FED 추적이라는 기능을 사용하는 것입니다. 추적은 (다양한 필터를 사용하는) 패킷 캡처 방법이며, FED에서 CPU에 푸시합니다. 그러나 FED 추적은 Cisco Catalyst 6500 시리즈 스위치의 Netdr 기능처럼 간단하지 않습니다.

다음은 이 프로세스를 여러 단계로 나눈 것입니다.

1. 상세정보 추적을 활성화합니다. 기본적으로 이벤트 추적은 켜져 있습니다. 실제 패킷을 캡처하기 위해 상세정보 추적을 활성화해야 합니다.

```
<#root>

3850-2#

set trace control fed-punject-detail enable
```

2. 캡처 버퍼를 미세 조정합니다. 상세정보 추적을 위한 버퍼의 깊이를 확인하고, 필요하다면 늘립니다.

```
<#root>

3850-2#

show mgmt-infra trace settings fed-punject-detail


One shot Trace Settings:

Buffer Name: fed-punject-detail
Default Size: 32768
Current Size: 32768
Traces Dropped due to internal error: No
Total Entries Written: 0
```

```
One shot mode: No
One shot and full: No
Disabled: False
```

다음 명령으로 버퍼 크기를 변경할 수 있습니다.

<#root>

3850-2#

set trace control fed-punject-detail buffer-size

사용 가능한 값은 다음과 같습니다.

<#root>

3850-2#

set trace control fed-punject-detail buffer-size ?

```
  <8192-67108864>  The new desired buffer size, in bytes
  default          Reset trace buffer size to default
```

3. 캡처 필터를 추가합니다. 이제 캡처를 위한 다양한 필터를 추가해야 합니다. 각기 다른 필터를 추가하고, 해당 캡처를 위해 이 필터를 전부 일치하거나 그중 하나를 일치하도록 선택합니다.

   다음 명령으로 필터를 추가합니다.

<#root>

3850-2#

set trace fed-punject-detail direction rx filter_add

현재 다음 옵션을 사용할 수 있습니다.

<#root>

3850-2#

set trace fed-punject-detail direction rx filter_add ?

```
  cpu-queue  rxq 0..31
  field      field
  offset     offset
```

이제 서로 연결해야 합니다. 이 문제 해결 프로세스의 2단계에서 식별한 문제의 원인인 대기열을 기억하십니까? 대기열 16이 CPU에 많은 패킷을 푸시하는 대기열이므로, 이 대기열을 추적하여 여기서 CPU에 펀트하는 패킷을 확인하는 것이 합당합니다.

다음 명령으로 임의의 대기열을 추적하도록 선택할 수 있습니다.

<#root>

3850-2#

set trace fed-punject-detail direction rx filter_add cpu-queue

이 예를 위한 명령은 다음과 같습니다.

<#root>

3850-2#

set trace fed-punject-detail direction rx filter_add cpu-queue 16 16

필터에 대해 match all 또는 match any 중 하나를 선택한 다음 추적을 활성화해야 합니다.

```
<#root>

3850-2#

set trace fed-punject-detail direction rx match_all


3850-2#

set trace fed-punject-detail direction rx filter_enable
```

4. 필터링된 패킷을 표시합니다. show mgmt-infra trace messages fed-punject-detail 명령으로 캡처한 패킷을 표시할 수 있습니다.

```
<#root>

3850-2#

show mgmt-infra trace messages fed-punject-detail


[11/25/13 07:05:53.814 UTC 2eb0c9 5661]
00 00 00 00 00 4e 00 40 07 00 02 08 00 00 51 3b
00 00 00 00 00 01 00 00 03 00 00 00 00 00 00 01
00 00 00 00 20 00 00 0e 00 00 00 00 00 01 00 74
00 00 00 04 00 54 41 02 00 00 00 00 00 00 00 00

[11/25/13 07:05:53.814 UTC 2eb0ca 5661]
ff ff ff ff ff ff aa bb cc dd 00 00 08 06 00 01
08 00 06 04 00 01 aa bb cc dd 00 00 c0 a8 01 0a
ff ff ff ff ff ff c0 a8 01 14 00 01 02 03 04 05
06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 f6 b9 10 32
[11/25/13 07:05:53.814 UTC 2eb0cb 5661] Frame descriptors:
[11/25/13 07:05:53.814 UTC 2eb0cc 5661]
=========
fdFormat=0x4    systemTtl=0xe
loadBalHash1=0x8        loadBalHash2=0x8
spanSessionMap=0x0      forwardingMode=0x0
destModIndex=0x0        skipIdIndex=0x4
srcGpn=0x54     qosLabel=0x41
srcCos=0x0      ingressTranslatedVlan=0x3
bpdu=0x0        spanHistory=0x0
sgt=0x0 fpeFirstHeaderType=0x0
srcVlan=0x1     rcpServiceId=0x2
wccpSkip=0x0    srcPortLeIndex=0xe
cryptoProtocol=0x0      debugTagId=0x0
vrfId=0x0       saIndex=0x0
pendingAfdLabel=0x0     destClient=0x1
appId=0x0       finalStationIndex=0x74
decryptSuccess=0x0      encryptSuccess=0x0
rcpMiscResults=0x0      stackedFdPresent=0x0
spanDirection=0x0       egressRedirect=0x0
redirectIndex=0x0       exceptionLabel=0x0
```

```
destGpn=0x0     inlineFd=0x0
suppressRefPtrUpdate=0x0        suppressRewriteSideEfects=0x0
cmi2=0x0        currentRi=0x1
currentDi=0x513b        dropIpUnreachable=0x0
srcZoneId=0x0   srcAsicId=0x0
originalDi=0x0  originalRi=0x0
srcL3IfIndex=0x2        dstL3IfIndex=0x0
dstVlan=0x0     frameLength=0x40
fdCrc=0x7       tunnelSpokeId=0x0

=========
[11/25/13 07:05:53.814 UTC 2eb0cd 5661]
[11/25/13 07:05:53.814 UTC 2eb0ce 5661] PUNT PATH (fed_punject_rx_process_packet:
830):RX: Q: 16, Tag: 65561
[11/25/13 07:05:53.814 UTC 2eb0cf 5661] PUNT PATH (fed_punject_get_physical_iif:
579):RX: Physical IIF-id 0x104d88000000033
[11/25/13 07:05:53.814 UTC 2eb0d0 5661] PUNT PATH (fed_punject_get_src_l3if_index:
434):RX: L3 IIF-id 0x101b6800000004f
[11/25/13 07:05:53.814 UTC 2eb0d1 5661] PUNT PATH (fed_punject_fd_2_pds_md:478):
RX: l2_logical_if = 0x0
[11/25/13 07:05:53.814 UTC 2eb0d2 5661] PUNT PATH (fed_punject_get_source_cos:638):
RX: Source Cos 0
[11/25/13 07:05:53.814 UTC 2eb0d3 5661] PUNT PATH (fed_punject_get_vrf_id:653):
RX: VRF-id 0
[11/25/13 07:05:53.814 UTC 2eb0d4 5661] PUNT PATH (fed_punject_get_src_zoneid:667):
RX: Zone-id 0
[11/25/13 07:05:53.814 UTC 2eb0d5 5661] PUNT PATH (fed_punject_fd_2_pds_md:518):
RX: get_src_zoneid failed
[11/25/13 07:05:53.814 UTC 2eb0d6 5661] PUNT PATH (fed_punject_get_acl_log_direction:
695): RX: : Invalid CMI2
[11/25/13 07:05:53.814 UTC 2eb0d7 5661] PUNT PATH (fed_punject_fd_2_pds_md:541):RX:
 get_acl_log_direction failed
[11/25/13 07:05:53.814 UTC 2eb0d8 5661] PUNT PATH (fed_punject_get_acl_full_direction:
724):RX: DI 0x513b ACL Full Direction 1
[11/25/13 07:05:53.814 UTC 2eb0d9 5661] PUNT PATH (fed_punject_get_source_sgt:446):
RX: Source SGT 0
[11/25/13 07:05:53.814 UTC 2eb0da 5661] PUNT PATH (fed_punject_get_first_header_type:680):
RX: FirstHeaderType 0
[11/25/13 07:05:53.814 UTC 2eb0db 5661] PUNT PATH (fed_punject_rx_process_packet:916):
RX: fed_punject_pds_send packet 0x1f00 to IOSd with tag 65561
[11/25/13 07:05:53.814 UTC 2eb0dc 5661] PUNT PATH (fed_punject_rx_process_packet:744):
RX: **** RX packet 0x2360 on qn 16, len 128 ****
[11/25/13 07:05:53.814 UTC 2eb0dd 5661]
buf_no 0 buf_len 128

<snip>
```

이 출력은 많은 정보를 제공하며 일반적으로 패킷이 어디에서 오며 여기에 포함된 내용을 검색하는 데 충분할 수 있습니다.

여기서도 헤더 덤프의 첫 부분은 시스템에서 사용하는 Meta-data입니다. 두 번째 부분은 실제 패킷입니다.

```
ff ff ff ff ff ff - destination MAC address
```

```
aa bb cc dd 00 00 - source MAC address
```

원인 포트를 찾기 위해 이 소스 MAC 주소를 추적하도록 선택할 수 있습니다(이 주소가 큐 16에서 펀팅되는 패킷의 대부분임을 식별한 경우, 이 출력은 패킷의 한 인스턴스만 표시하고 다른 출력/패킷은 클리핑됩니다).

하지만 더 나은 방법이 있습니다. 헤더 정보 다음에 오는 로그를 확인합니다.

```
[11/25/13 07:05:53.814 UTC 2eb0ce 5661] PUNT PATH (fed_punject_rx_process_packet:
830):RX: Q: 16, Tag: 65561
[11/25/13 07:05:53.814 UTC 2eb0cf 5661] PUNT PATH (fed_punject_get_physical_iif:
579):RX: Physical IIF-id 0x104d88000000033
```

첫 번째 로그는 이 패킷이 어떤 대기열 및 태그에서 오는지를 명확히 알려줍니다. 이전에 대기열을 인식하지 못한 경우 어떤 대기열이었는지를 쉽게 식별할 수 있는 방법입니다.

두 번째 로그는 소스 인터페이스에 대한 물리적 IIF(Interface ID Factory)-ID를 제공하므로 더욱 유용합니다. 16진수 값은 포트에 관한 정보를 덤프하는 데 사용할 수 있는 핸들입니다.

<#root>

```
3850-2#
```

show platform port-asic ifm iif-id 0x0104d88000000033

```
Interface Table
Interface IIF-ID        : 0x0104d88000000033
Interface Name          : Gi2/0/20
Interface Block Pointer : 0x514d2f70
Interface State         : READY
Interface Stauts        : IFM-ADD-RCVD, FFM-ADD-RCVD
Interface Ref-Cnt       : 6
Interface Epoch         : 0
Interface Type          : ETHER
        Port Type         : SWITCH PORT
        Port Location     : LOCAL
        Slot              : 2
        Unit              : 20
        Slot Unit         : 20
        Active            : Y
        SNMP IF Index     : 22
        GPN               : 84
        EC Channel        : 0
        EC Index          : 0
        ASIC              : 0
        ASIC Port         : 14
        Port LE Handle    : 0x514cd990
Non Zero Feature Ref Counts
```

```
            FID : 48(AL_FID_L2_PM), Ref Count : 1
            FID : 77(AL_FID_STATS), Ref Count : 1
            FID : 51(AL_FID_L2_MATM), Ref Count : 1
            FID : 13(AL_FID_SC), Ref Count : 1
            FID : 26(AL_FID_QOS), Ref Count : 1
    Sub block information
            FID : 48(AL_FID_L2_PM), Private Data &colon; 0x54072618
            FID : 26(AL_FID_QOS), Private Data &colon; 0x514d31b8
```

소스 인터페이스 및 원인을 다시 한 번 확인했습니다.

추적은 CPU 사용량이 많은 문제를 해결하는 데 매우 중요하며, 이러한 상황을 성공적으로 해결하기 위해 많은 정보를 제공하는 강력한 도구입니다.

Cisco Catalyst 3850 Series Switch용 EEM(Embedded Event Manager) 스크립트 샘플링

특정 임계값에서 로그를 생성하도록 트리거하려면 다음 명령을 사용합니다.

```
process cpu threshold type total rising


       interval


switch
```

이 명령으로 다음과 같은 로그가 생성됩니다.

```
*Jan 13 00:03:00.271: %CPUMEM-5-RISING_THRESHOLD: 1 CPUMEMd[6300]:  Threshold: : 50, Total CPU Utilzati
```

생성된 로그는 다음 정보를 제공합니다.

- 트리거 시점의 총 CPU 사용률. 이 예에서는 Total CPU Utilization(total/Intr):50/0으로 식별됩니다.
- 상위 프로세스 - PID/CPU%의 형식으로 표시됩니다. 이 예에서는 다음과 같습니다.

```
    8622/25 - 8622 is PID for IOSd and 25 implies that this process is using 25% CPU.
```

```
5753/12 - 5733 is PID for FED and 12 implies that this process is using 12% CPU.
```

EEM 스크립트는 여기에 표시됩니다.

```
event manager applet highcpu
event syslog pattern "%CPUMEM-5-RISING_THRESHOLD"
 action 0.1 syslog msg "high CPU detected"
 action 0.2 cli command "enable"
 action 0.3 cli command "show process cpu sorted | append nvram:<filename>.txt"
 action 0.4 cli command "show process cpu detailed process <process name|process ID>
 sorted | nvram:<filename>.txt"
 action 0.5 cli command "show platform punt statistics port-asic 0 cpuq -1
 direction rx | append nvram:<filename>.txt"
 action 0.6 cli command "show platform punt statistics port-asic 1 cpuq -1
 direction rx | append nvram:<filename>.txt"
 action 0.7 cli command "conf t"
 action 0.8 cli command "no event manager applet highcpu"
```

참고: process cpu threshold 명령은 현재 3.2.X 열차에서 작동하지 않습니다. 또 다른 기억해야 할 점은 이 명령이 4개 코어 사이의 평균 CPU 사용률을 확인하고 이 평균이 명령에 정의된 백분율에 도달할 때 로그를 생성한다는 것입니다.

Cisco IOS XE 16.x 이상 릴리스

Cisco IOS® XE Software Release 16.x 이상을 실행하는 Catalyst 3850 스위치가 있는 경우 IOS-XE 16.x를 실행하는 Catalyst 스위치 플랫폼의 높은 CPU 사용량 문제 해결을 참조하십시오.

관련 정보

- Cisco IOS XE란?
- Cisco Catalyst 3850 스위치 - 데이터 시트 및 문서
- Cisco 기술 지원 및 다운로드

이 번역에 관하여

Cisco는 전 세계 사용자에게 다양한 언어로 지원 콘텐츠를 제공하기 위해 기계 번역 기술과 수작업 번역을 병행하여 이 문서를 번역했습니다. 아무리 품질이 높은 기계 번역이라도 전문 번역가의 번역 결과물만큼 정확하지는 않습니다. Cisco Systems, Inc.는 이 같은 번역에 대해 어떠한 책임도 지지 않으며 항상 원본 영문 문서(링크 제공됨)를 참조할 것을 권장합니다.